ITOS Telemetry & Command Interfaces Guide

Integrated Test & Operations System
$Date: 2006/09/25 22:04:12 $

ITOS Development & Support Group

NASA/GSFC Code 584, Greenbelt MD 20771

Copyright 1999-2006, United States Government as represented by the Administrator of the National Aeronautics and Space Administration. No copyright is claimed in the United States under Title 17, U.S. Code.

This software and documentation are controlled exports and may only be released to U.S. Citizens and appropriate Permanent Residents in the United States. If you have any questions with respect to this constraint contact the GSFC center export administrator, <Thomas.R.Weisz@nasa.gov>.

This product contains software from the Integrated Test and Operations System (ITOS), a satellite ground data system developed at the Goddard Space Flight Center in Greenbelt MD. See <http://itos.gsfc.nasa.gov/> or e-mail <itos@itos.gsfc.nasa.gov> for additional information.

You may use this software for any purpose provided you agree to the following terms and conditions:

1. Redistributions of source code must retain the above copyright notice and this list of conditions.
2. Redistributions in binary form must reproduce the above copyright notice and this list of conditions in the documentation and/or other materials provided with the distribution.
3. All advertising materials mentioning features or use of this software must display the following acknowledgement:

This product contains software from the Integrated Test and Operations System (ITOS), a satellite ground data system developed at the Goddard Space Flight Center in Greenbelt MD.

This software is provided ‘‘as is’’ without any warranty of any kind, either express, implied, or statutory, including, but not limited to, any warranty that the software will conform to specification, any implied warranties of merchantability, fitness for a particular purpose, and freedom from infringement and any warranty that the documentation will conform to their program or will be error free.

In no event shall NASA be liable for any damages, including, but not limited to, direct, indirect, special or consequential damages, arising out of, resulting from, or in any way connected with this software, whether or not based upon warranty, contract, tort, or otherwise, whether or not injury was sustained by persons or property or otherwise, and whether or not loss was sustained from or arose out of the results of, or use of, their software or services provided hereunder.

Introduction

The ITOS's reason for being is to ingest telemetry and send commands. This document presents the interfaces ITOS presents to the outside world for accomplishing those tasks.

The primary mechanism for both transmitting commands and receiving telemetry is an IP network. The system can communicate using TCP or UDP. It can initiate or accept TCP connections, and can receive and transmit UDP unicast or multicast datagrams.

For telemetry, the system was designed to accept CCSDS version 1 transfer frames or packets. It also can process other forms of telemetry, including TDM, with certain restrictions. It can handle a number of telemetry headers and trailers, and can perform CRC and Reed-Solomon error control.

For commands, the system transmits CCSDS telecommand transfer frames, which optionally may be coded into CCSDS command link transmission units. It can generate a number of command headers and trailers, encapsulate commands in NASCOM 4800-bit blocks, and generate ASCII-hex data for output to a SCAT-based front-end computer.

The system also can perform command throttling, inserting a programmable delay between transmitting individual commands in a sequence.

The system is designed to provide a great deal of flexibility as delivered, and to be easily modified to provide support for additional command and telemetry types and wrappers.

1 Telemetry Types

A program with the unfortunate name `frame_sorter` is the primary telemetry interface in the ITOS. It accepts several different kinds of telemetry, which it can sort into archive files or output queues destined for other programs and further processing. It also can extract CCSDS source packets from a transfer frame stream and sort them by application ID into archives or output queues.

The system can run multiple `frame_sorter` processes and thus can process multiple telemetry streams simultanseously. That, for example, is how we handle both live spacecraft telemetry and ground station monitor blocks at the same time. In the SMEX-Lite integration and test environment, we receive spacecraft telemetry, and CCSDS-formatted telemetry from the power GSE and the attitude control system's dynamic simulator.

1.1 Supported Telemetry Formats

The ITOS accepts the following types of telemetry, some of which were given misleading names:

'`ccsds`' These are version 1 telemetry frames, called transfer frames. This input requires three arguments: frame length in bytes, CCSDS version number, and spacecraft ID. Obviously, the version always should be zero. The length should be the length of the transfer frame proper from the start of the primary header through the end of the Operational Control Field (usually a CLCW), if present, but not including the attached sync marker or any frame error control. Sync markers and error control fields must be handled as wrappers.

'`aos`' These are version 2 telemetry frames, called virtual channel data units (VCDUs). This input requires four arguments: VCDU length in bytes, VCDU Insert Zone length in bytes, spacecraft ID, and OCF flag. The VCDU length should be the length of the transfer frame proper from the start of the primary header through the end of the Operation Control Field (OCF), if present, but not including the attached sync marker or any frame error control. Sync markers and error control fields must be handled as wrappers. This implementation assumes that every VCDU contains packets.

'`packet`' These are CCSDS telemetry source packets, common to both version 1 and 2 CCSDS telemetry.

'`nascom`' These are 4800-BIT nascom blocks in cases where we expect to extract data directly from the data field. This is used by the SMEX project to extract data from ground station monitor blocks.

'`spartan`' These are TDM major frames. We developed the capability to process TDM data for the Spartan project; hence, the name. There is code specific to Spartan telemetry which sorts the archives and output by the major frame counter, but a similar, generalized PCM processor easily could be constructed from this base.

'`itp`' This is a very specialized input: It processes CCSDS source packets inside the ITP wrapper (which we'll discuss later). The ITP message may contain multiple packets, and the `frame_sorter` was designed to deal with a single data item at a time, with multiple wrappers, not the reverse. This input is used to accept data from GSFC's old Code 521's Front-end Telemetry & Command Processor (FTCP).

'`repack`' This is a generally useful input: It accepts input blocks of a given size, and extracts output messages of a different size, aggregating or splitting the input as needed.

1.2 Repack Input Notes

This input requires three arguments: input frame length, output frame length, and the sync pattern expected on the output frames. See the note below.

The '`repack`' input is very useful, especially with the '`junk`' wrapper, which can be used to skip over anything at the beginning or end of in input frame. If the input wrappers contain a sequence count, the repack input looks for sequence continuity and will try to resynchronize to the input if it sees a sequence error.

If the sync pattern argument is non-zero, the '`repack`' will check that the given sync is present on every frame it is about to output. It uses the length of the given sync to determine how many bytes of the ouput it should compare. If the sync marker is not present, it will scan the last input looking for it, and if it still doesn't find it, it will continue inputting frames and searching until it finds an output sync. Then it will pick things up as usual.

2 Telemetry Wrappers

Telemetry can come in a variety of headers and trailers, and the system is architected to handle these flexibly. It incorporates an extensible wrapper library, `libtmw`, which is used by many telemetry processing programs to handle telemetry wrappers.

Wrappers are specified to the system as a list of names, and are processed from left to right, outermost to innermost. Some wrappers take optional positional arguments, which are separated from the wrapper name by a colon and separated from each other with commas.

2.1 Wrapper Data

The following data may be extracted from wrappers or set by a wrapper processor.

```
    unsigned wrapper_errors;           /* non-zero if wrapper error seen   */
    enum boolean crc_enab; /* true if CRC checking enabled     */
    enum boolean crc_error; /* true if CRC error detected     */
    enum boolean sync_in_crc; /* true if sync in CRC calculation  */
    enum boolean reversed; /* true if bits in reversed order   */
    enum boolean inverted; /* true if bits appear inverted     */
    enum boolean last_frame; /* true if last frame in session    */
    unsigned sync_error_bit_count; /* 0-15     */
    enum boolean frame_bit_slip; /* true if bit slip detected     */
    FrameSyncState sync_state; /* frame sync state for this frame  */
    enum boolean rs_enab; /* Reed-Solomon ECC enabled     */
    enum boolean rs_error; /* Reed-Solomon uncorrectable error */
    enum boolean rs_corrected; /* Reed-Solomon corrected errors    */
    enum boolean mc_seq_chk_enab; /* master channel sequence chk enab */
    enum boolean mc_seq_error; /* master channel sequence error    */
    unsigned msg_length; /* msg (frame) length from wrapper  */
/* (must be total message length)   */
    UNIX_TIME rcpt_time; /* time frame received by front-end */
    unsigned pkt_count; /* count of packets in ITP message  */
    unsigned seq_num; /* sequence count of message     */
    unsigned seq_mod; /* modulus of sequenct count     */
```

2.2 Supported Wrappers

These are the currently recognized wrappers:

‘anno12’ This is the 12-byte annotation that the `frame_sorter` and the old FTCP prepends to CCSDS source packets extracted from a transfer frame stream. By default, it gets put on all derived output from the `frame_sorter` to carry forward some of the information extracted from the frame layer and ground message wrappers.

‘anno12aos’

This is the 12-byte annotation that the `frame_sorter` prepends to CCSDS source packets extracted from an AOS VCDU stream. It is identical in format and function to the ‘`anno12`’ header, except that its first two bytes are copied from the first two bytes of the header of the VCDU which contained the annotated source packet.

‘`ccsdstf`’ This is used only by the ITOS packet dump facility for formatting the header information in its output.

‘`crc`’ This is a CRC-16 error detector. This wrapper takes two arguments: the number of bytes to skip before beginning the CRC calculation, and the polynomial to use in the calculation. These default, respectively, to zero and the CCSDS recommended polynomial 0x11021 (`x^16 + x^12 + x^5 + 1`). This wrappers strips off the 2-byte trailer which is the CRC check word.

‘`crcword`’ This is a no-op 2-byte trailer, intended for skipping the CRC word when it’s been processed by something upstream, but not removed from the frame. You could just as well use ‘`junk:0,2`’ for this.

‘`ddd`’ This is the Deep Space Network (DSN) Data Delivery (DDD) header.

‘`fep521`’ This is an obsolete header we added for the old GSFC Code 521.

‘`ftcp`’ This is the ITP header with an 8-byte trailer, and is the wrapper used for transfer frames by the FTCP.

‘`itp`’ This is the 16-byte ITP header that gets put on the front of a lot of our telemetry for mostly historical reasons.

‘`junk`’ This is a way to skip useless header and/or trailer data, and takes two arguments: a header length and a trailer length, both of which default to zero. This is particularly useful with the `repack` input, and for skipping attached sync markers and error control fields.

‘`rs`’ This is the Reed-Solomon decoder, and it takes two arguments: The interleave factor, `I`, from 1 to 5; and the virtual fill length, VF, in bytes, divided by `I`. The defaults are 5 and 0.

‘`rsdebug`’ This is the same as ‘`rs`’, but it prints warnings to the event log for each frame in which it finds a correctable or uncorrectable error.

‘`smex`’ This is the SMEX telemetry header, used by all SMEX missions after FAST. It is better known as the LEO-T header.

‘`tsifep`’ This is a collection of TSI Telsys Inc. front-end processor (FEP) trailers: `tsifep_rsqat`, `tsifep_tct`, and `tsifep_fsqt` in order outermost to innermost. This takes two optional arguments: The first selects the wrapper collection, the second is passed to the `tsifep_tct` wrapper. If the first argument to `tsifep` is `tdm`, the `tsifep_rsqat` trailer is not included in the wrapper collection; if the argument is `ccsds` or not given, all three trailers are included.

‘`tsifep_fsqt`’

This is the 2-byte TSI FEP frame synchronization quality trailer.

‘tsifep_tct’

This is the 10-byte TSI FEP time code trailer. The time code may be of two forms, selected by optional argument: cds, the default, selects the CDS time code; pb4 selects the PB4 time code.

‘tsifep_rsqat’

This is the 32-byte TSI FEP Reed-Solomon quality annotation trailer.

‘tsisp_ah’

This is the 24-byte TSI service processor "Gem" annotation header.

2.2.1 12-byte Annotation Header Formats

An ‘anno12’ 12-byte annotation header looks like this:

```
             +------+-----+-----+
             | enab | err | cor |
              \.....       ..../
                    \      /
+----+------+----+---+----+---+-------+-------+-------+------------+
| vn | scid | vc | ? | rs | ? | t_fmt | flags |  fill |  gnd_time  |
| 2  |  10  | 3  | 1 | 3  | 1 |   4   |   8   |  16   |     48     |
+----+------+----+---+----+---+-------+-------+-------+------------+
| [0]  |   [1]       |    [2]          |  [3]  |[4]|[5]|[6]|...|[11]|
                                     /         \
             ........................           .........
           /                                             \
          | pkt | dir | pkt | crc | crc | inc | vc  | tf  |
          | hdr |     | seq | err | ena | pkt | seq | hdr |
          | err |     | err |     |     |     | err | err |
          +-----+-----+-----+-----+-----+-----+-----+-----+
          |                      [3]                      |
```

The ‘anno12aos’ header is identical, except the first two bytes have the following format:

```
+----+------+----+
| vn | scid | vc |
| 2  |   8  | 6  |
+----+------+----+
| [0]  |   [1]   |
```

The fields are, in order:

field	word(s)	bit(s)	description
frame version	0	0-1	version field from CCSDS transfer frame hdr.
frame s/c ID	0	2-11	spacecraft ID from CCSDS transfer frame hdr.
frame VC ID	0	12-14	virtual channel ID from CCSDS transfer frame hdr.
reserved	0	15	

Reed-Solomon enabled	1	0	if set, Reed-Solomon error detection and correction enabled.
Reed-Solomon error	1	1	if set, uncorrectable Reed-Solomon error(s) encountered.
Reed-Solomon corrected	1	2	if set, the Reed-Solomon code corrected one or more errors.
reserved	1	3	
time format	1	4-7	Defines time code format. Acceptable values are given in the next table.
packet header error	1	8	if set, packet header extracted from frame with uncorrectable error.
data direction reversed	1	9	if set, data received in reverse bit order.
packet sequence error	1	10	if set, this packet's sequence count is not the successor or the previous packet with the same application ID on the same VC.
frame error	1	11	if set, uncorrectable error detected in one or more frames from which this packet was extracted.
frame error enabled	1		12 if set, frame error checking was enabled.
incomplete packet	1	13	if set, packet is incomplete, and filled to it's indicated length beginning at ‘`fill location`’.
VC sequence error	1	14	if set, a transfer frame from which this packet was extracted was not the successor of the previous frame on the same virtual channel.
frame header error	1	15	a frame from which this packet was extracted had an incorrect version or spacecraft ID.
fill location	2	0-15	byte offset from the end of the packet primary header of packet fill data, if ‘`incomplete packet`’ is set.

ground received time	4-6	0-15	ground received time extracted from frame wrappers in format defined by '`time format`' above.

In the above table, '`words`' are 16-bits, and bits are in CCSDS order; that is, bit 0 is the most significant bit.

Time format codes are:

'`0`'	none
'`1`'	PB1 code
'`2-3`'	reserved
'`4`'	PB4 code
'`5-7`'	reserved
'`8`'	relative TIME42, a time in CCSDS Unsegmented Code (CUC)
'`9`'	absolute TIME42, a date in CUC, default for anno12 headers created by ITOS
'`10-15`'	reserved

2.2.2 ITP Header Format

The ITP header is defined as follows:

Field	Bytes	Description
message length	0-1	message length, including ITP header.
message class	2-3	message data class; for example: telemetry, command, etc.
message type	4-5	
signature	6-7	
message subtype	8-11	
PDU count	12-13	in some contexts, the number of packet data units in the message.
user area	14-15	

2.2.3 SMEX / LEO-T Telemetry Header Format

```
A SMEX / LEO-T telemetry header looks like:

   +----+-----+-------+-------+
   | vn | len | flags | time  |
   | 2  | 14  |   16  |  48
   +----+-----+-------+-------+
             /         \
      ......            .........................................
     /                                                          \
    | rs  | rs  | crc | crc | mcs | mcs | inv | frm  | rev | class |
| ena | err | ena | err | chk | err |     | sync |    |       |
|     |     |     |     | ena |     |     | mode |    |       |
|  1  |  1  |  1  |  1  |  1  |  1  |  2  |  2   |  1 |   5   |
+-----+-----+-----+-----+-----+-----+-----+------+----+-------+
```

where the fields are, in order:

field	word(s)	bit(s)	description
header version	0	0-1	header version number
message length	0	2-15	length of the message including the 10-byte SMEX header.
flags	1	0-15	Annotation flags.
Reed-Solomon enabled	1	0	Reed-Solomon decoding was performed by the ground station.
Reed-Solomon error	1	1	Message contains uncorrectable Reed-Solomon errors.
CRC enabled	1	2	Cyclical Redundancy Checking was performed by the ground station.
CRC error	1	3	The CRC indicated errors in the message.
MCS check enabled	1	4	Master channel sequence number checking was performed by the ground station.
MCS error	1	5	A master channel sequence discontinuity was seen at the station.
inversion flags	1	6-7	0, data true; 1, data inverted; 2, inverted data made true.
frame sync mode	1	8-9	0, search; 1, check; 2, lock; 3, flywheel.
reverse data	1	10	Data bits reversed.
data class	1	11-15	1, spacecraft telemetry; 2, spacecraft command; 3, tracking data; others undefined.
ground received time	2-4	0-47	Ground received time stamp, in PB5 format.
PB5 flag bit	2	0	Always zero.
PB5 truncated Julian day	2	1-14	Unsigned count of days from the epoch which began October 10, 1995.
PB5 seconds of day	2	15	Seconds of day, most significant bit.
PB5 seconds of day	3	0-15	Seconds of day, remaining bits.
PB5 milliseconds	4	0-9	Milliseconds of day.

Spare	4	10-15	Unused bits.

2.3 End-of-Session Indication

Telemetry sources may indicate end-of-session to ITOS by sending an empty outermost telemetry header; that is, a header that indicates by it's length field that no data follows. (*It is not possbile to send an end-of-session indication with an outermost header that does not carry a message length field.*)

ITOS does not archive end-of-session messages, but it does pass them to programs receiving telemetry streams. Applications receiving a CCSDS packet stream from ITOS will be sent an empty ITP header to indicate end-of-session; that is, an ITP header with its length field set to the ITP header length (16). Applications receiving from ITOS a copy of ITOS's telemetry input will receive the same end-of-session token that ITOS received.

ITOS may be set up to shut down a telemetry stream upon receiving end-of-file from the telemetry source. If this is done, then ITOS will generate end-of-session tokens on all telemetry outputs before shutting down the telemtry stream.

3 Telemetry Networking

The frame_sorter can accept telemetry using Internet Protocol (IP) sockets, over either the Transport Control Protocol (TCP) or User Datagram Protocol (UDP). It can acccept or initiate TCP connections, or receive unicast or multicast UDP datagrams.

In 'server_tcp' mode, the system will listen on a given IP port for a connection. When the connection is broken, the program will reset and begin listening for a new connection. This cycle will continue until end-of-session or until the input is explicitly destroyed.

In 'client_tcp' mode, the system will initiate a connection to a given host and port combination. If the connection is broken, the input is destroyed.

In 'udp' mode, the system will create a socket at a given port number and read datagrams. If given a host address in UDP, and that address is a class D address, the frame_sorter program will join the given multicast group automatically.

The manner in which the frame_sorter is set up is controlled by the telemetry controller.

4 Telemetry Frame Synchronization

The ITOS comes with a frame synchronization program, which can do bitwise frame sync on serial or IP data and data in disk files that contain no extra wrappers just raw telemetry.

The `FrameSync` program is capable of syncronizing frames having up to 32 bits of sync into output frame words of up to 16 bits placed into a specified number of 8 bit bytes per word. The program can be set up to work on input bytes that are in reverse bit order (from lsb to msb). Sync detection for inverted (one's compliment) data is automatic and inverted data will be automatically inverted back to normal before being output. The program has been tested with an EDT SSE and CDA-16 serial interface boards at input rates up to 40 Mbits/sec in the worse case condition with raw logging enabled and 50 Mbits/src without. Higher rates are attainable with faster multiple processor computers.

The input data is byte oriented stream that can come from a serial input device, an IP stream using UDP or TCP as a server or a client. Data can also be input from a file to act like a playback for testing. The input data stream can be optionally archived to disk while being processed for later recovery.

The output data frames can have optional headers and/or trailers appended as well as optional deradomization applied. The output stream connection can be as a TCP/IP server, UDP, or an output pipe.

The program is a state machine of SEARCH, CHECK, LOCK, & FLY-WHEEL states.

```
   ----------      ----------      ----------      ----------
   |        |      |        |      |        |----->|        |
-->| SEARCH |----->| CHECK  |----->|  LOCK  |      |FLYWHEEL|
   |        |      |        |      |        |<-----|        |
   ----------      ----------      ----------      ----------
        ^                                                |
        |                                                |
        --------------------------------------------------
```

`SEARCH`
: Program searches for frame sync pattern until it finds it either normal or inverted then changes to CHECK state.

`CHECK`
: Program checks specified number of successive frames for sync in the proper place before declaring a LOCK condition. If this condition fails then program returns to SEARCH state and all check frames are discarded. If sync is maintained all checked frames are transmitted and the program goes to LOCK state.

`LOCK`
: Program is now aligned on the frame sync pattern and will continue transferring data to the output as telemetry frames of predetermined size. Sync (normal and inverted) will be continued to be tested on incoming frames. Program will remain in this state unless sync is lost when it goes to FLY WHEEL state. The program will allow for forward sync slippage (1 or more extra bits before sync) specified from 1 to sync word size in bits before declaring and out of lock condition. No backward bit slippage (1 or more missing bits) is allowed by the program. This is an automatic drop lock condition. If sync is found to be inverted, the succeeding frame data will be automatically inverted to match.

`‘FLYWHEEL’`

If sync is lost on the input for up to specified FLY WHEEL value number of frames then program will continue to transfer frames to the output. If Fly wheel count is exceeded and sync is not found in the proper position then program drops back into SEARCH mode. If sync is found then program returns to LOCK state.

Status mnemonics are available in the database for monitoring

```
lock status GBL_FRMSYNC_STAT,
good frames received count GBL_FRMSYNC_CNT,
drop lock count GBL_FRMSYNC_DROP,
total bytes received GBL_FRMSYNC_BYTES,
frames per second GBL_FRMSYNC_FPS,
source inversion state GBL_FRMSYNC_INV,
total all bytes received GBL_FRMSYNC_BYTES and
total bytes received in search mode GBL_FRMSYNC_SBYTES.

Additional mnemonics for the EDT board interace are
EDT channel selected GBL_FRMSYNC_CHAN and
EDT buffer overrun count GBL_FRMSYNC_OVERRUN.
EDT rate change GBL_FRMSYNC_RATE allows the user to change the
buffering of the frame sync program based on the rate changed
through this database variable. Data flow will be momentarily
disrupted during the reconfiguration and then restarted.
Valid values are 100 to 400M bits per seconds.
```

Caveats for running FrameSync program on RedHat Linux:

The FrameSync program is built against the EDT library ‘`libedt.so`’. RedHat does not support the Lazyload feature in it’s loader which will cause the program to fault and exit if the EDT driver is not loaded on the system running ITOS. To get around this problem for users wanting to use FrameSync without an EDT we have included the ‘`libedt.so`’ that FrameSync was built against in the ‘`$ITOS_DIR/etc`’ directory. The user can then create or add to their local shell environment *LD_LIBRARY_PATH* with ‘`$ITOS_DIR/etc`’ directory in the search path in your shell startup script like ‘`.bashrc`’ or ‘`.cshrc`’.

```
For ‘csh’ like shells use something like:
    setenv LD_LIBRARY_PATH <ITOS_DIR>/etc
or
    setenv LD_LIBRARY_PATH $LD_LIBRARY_PATH:<ITOS_DIR>/etc

For ‘sh’ like shells use something like:
    export LD_LIBRARY_PATH=<ITOS_DIR>/etc
or
    export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:<ITOS_DIR>/etc

where <ITOS_DIR> is replaced with the directory where the ITOS
package was installed.
```

Command line options include the following:

```
FrameSync <options>
```

```
where <options> are:
  -sync,  -fbits, -sbits,  -wbits,  -bytes,   -slip,   -check,   -fly,
  -port,  -file,  -fdelay, -server, -client,  -iport,  -comm,    -baud,
  -stop,  -data,  -parity, -hwhand, -verbose, -silent, -reverse, -drand,
  -retry, -timeo, -fifo,   -help,   -debug,   -debug1, -debug2,  -debug3,
  -wrap,  -edt,   -unit,   -chan,   -chan2,   -brate

In options below, Uppercase letters indicate minimum characters required.

** General Options **
  -BYtes <count>   -- number of bytes in output frame word > 0, default = 2
  -CHEck <count>   -- number of sync'd frames in a row before lock state
                      is achieved. If sync is not maintained for this many
                      frames all checked frames will be discarded and the
                      program will return to search state
                      0 means no checking is done, default = 0
  -DEBUG           -- sets verbosity to debug level, 4 or 4++, see -verbose
  -DEBUG1          -- sets verbosity to debug level, 5, see -verbose
  -DEBUG2          -- sets verbosity to debug level, 6, see -verbose
  -DEBUG3          -- sets verbosity to debug level, 7, see -verbose
  -DRand           -- frame sync will derandomize incoming frames
  -FBits <bits>    -- input frame size (including sync) in bits > 0
  -FLy   <count>   -- number of out of sync frames before drop lock,
                      default = 0
  -HELP or -?      -- Print this usage message and exits
  -RAw   <string>  -- optional filename to save raw input,
                      date and time are appended to the filename,
                      if no filename given then default = FrameRAW
  -RLen  <number>  -- optional raw file size limit before opening a,
                      new one and appending '.xxx' to filename,
                      Size is in MBytes not to exceed 2000, default = 100,
  -RETry           -- Try to reconnect to the input after EOF rather than
                      exiting.  If the reconnect fails then exit. This is
                      usefull for socket or serial connections that might
                      get extraneous hangups such as with -hwhand option
  -REVerse         -- reverse order bits in word msb->lsb ... lsb->msb
  -SIlent          -- output no debug messages, sets verbosity level to 0
  -SYnc  <pattern> -- defines sync pattern, may use 0x notation for hex
  -SBits <bits>    -- sync size in bits (1 to 32)
  -SLip  <bits>    -- allowable sync slippage in bits, default = 0
  -TImeo <msecs>   -- number of milliseconds ( > 0) after input data stops
                      that is considered Loss-Of-Data, 0 for no timeout,
                      default = 10000 (or 10 seconds)
  -VERBOSE         -- increase vebosity level for debug messages,
                      each -verbose adds 1 to verbosity level, default = 1,
                      Levels above 3 should only be used for debugging:
                      0 -- silent, nothing except error and panic messages
                      1 -- normal, + some warning messages
                      2 -- verbose, all warnings and some info messages
                      3 -- very verbose, all warnings and all info messages
                      4 -- debug,  + I/O config & periodic status message
                      5 -- debug1, + status message for every frame
                      6 -- debug2, + frame dump to STDERR in ASCII
                      7 -- debug3, + termio dump & sync mask config data
  -WBits <bits>    -- frame word size in bits (1 to 16), default = 16

** Input Options - File **
  -FDelay <usecs>  -- Delay between frame outputs when -file used,
                      default = 0
  -FILe  <name>    -- name of file to use as input instead of -comm,
                      default = NULL

** Input Options - EDT Serial **
  -EDT   <string>  -- optional input EDT device as input instead of -comm,
                      default = 'edt'
                      -file, -iport, -comm and -edt are mutually exclusive
  -UNit  <number>  -- Unit number of EDT device, default = 0
  -CHAn  <number>  -- Channel number, default = 0
  -CHAN2 <number>  -- Alternate Channel number if any, not same as -chan,
                      default = -1
  -BRate <number>  -- Expected EDT thruput rate in bits/sec, can use
                      shorthand notation of K for kilo or M for mega,
                      Valid range is 100 - 400M,
                      If then option is not given the program will look at
                      GBL_FRMSYNC_RATE mnemonic for the bit rate in bps.
                      If this is out of range then the default = 1M

** Input Options - Serial **
  -BAud  <string>  -- input baud rate, 300 .. 38400,...  default = 19200
  -COmm  <string>  -- input comm port to receive data, default = NULL
                      -file, -iport, -comm and -edt are mutually exclusive
                      If none are specified, data input from STDIN
  -DAta   <string> -- number of data bits (7 or 8), default = 8
  -PArity <string> -- parity ('even', 'odd', 'none'), default = 'none'
  -STop   <string> -- number of stop bits (1 | 2), default = 1
  -HWhand          -- enable hardware handshaking (RTS/CTS & DSR/DTR),
                      with this enabled a modem disconnect will cause an
                      EOF on the input which will cause the program to exit
                      unless the -retry option is enabled

** Input Options - Socket **
  -CLient <string> -- input is to be a socket client either UDP or TCP,
                      default = NULL
                      -server & -client are mutually exclusive,
  -IPort  <string> -- Used with -server or -client, this defines the
                      host and port used for input, default = NULL
```

```
                          When -client format is host:port,
                          If no -server or -client defaults to -server TCP
      -SErver <string> -- input is to be a socket server either UDP or TCP,
                          default = NULL
                          -server & -client are mutually exclusive

    ** Output Options **
      -FIFo  <name>    -- Output goes to a named FIFO. default = NULL,
                          -fifo & -port are mutually exclusive,
      -POrt  <number>  -- port number to listen for output socket connection,
                          default = 0, no port assigned, no data output,
                          unless -silent & no -fifo then output goes to STDOUT
      -PRoto           -- output socket protocol, TCP or UDP, default = TCP
      -WRap  <list>    -- Quoted space seperated list of header/trailer
                          wrappers output with frames, default = NULL,
                          See ITOS T&C Interfaces - telemetry wrappers,
                          The most usefull header is 'smex' a.k.a. LEO-T.
```

5 Telemetry Output

The `frame_sorter` program can provide a telemetry stream to customer ground support equipment (GSE) computers. In all cases, it can provide a copy of it's input stream, and in most cases, it can provide a subset of the input stream.

To request telemetry output, applications should initiate a TCP connection to the ITOS telemtry controller on the main ITOS workstation. The controller normally listens on port 32000, but the port number, is configurable at run time. Once a connection is formed, request data by issuing an ASCII acquire command of the form

`ac` *transport* `[`*destination*`]` *flowtype* `[`*filter*`]`

The *transport* argument may be one of '`client_tcp`', '`server_tcp`' (or just '`tcp`'), or '`udp`' depending on which transport the user desires. The *destination* which may be specified for each *transport* is given by the following table. Note that IP addresses may be substituted for host names.

'`client_tcp`'

for client-side TCP/IP. This is followed by a set of hostname and port number pairs and other parameters controlling failover and retry. It has the form:

`client_tcp [`*cycles*`]` *host port* `[`*host port* `[`*host port*`]] [`*retries* `[`*interval*`]]`

where:

cycles is the number of times ITOS should cycle through the list of host/port pairs trying to establish or re-establish a connection before giving up.

host
port are a hostname and port number to which the ITOS should initiate a connection.

retries is the number of times ITOS should try initiating a connection to each host/port pair in the list.

interval is the time in seconds which ITOS should wait between retries.

'`server_tcp`'

or simply '`tcp`', for server-side TCP/IP. This is followed by an optional *port* number on which the ITOS should listen for a connection, followed by an optional *interval* in seconds that ITOS should wait for a connection request to come in. Once established, if the connection is broken, ITOS will return to listening for another connection for up to the given interval. The default interval is 30 seconds; a zero interval means ITOS should wait forever.

'`udp`' for UDP. This is followed by a *host* name and *port* number to which the ITOS will send data.

The *flowtype* may be one of '`frames`' or '`pkts`' for acquiring transfer frames or source packets, respectively. If frames have been requested, the *filter* option is a list of VCIDs of the form '`vcX`', where '`X`' is a digit from 0 to 7. If packets have been requested, the *filter* option is of the form '`vcX [`*apid* `[`*apid*`...]]`', where '`vcX`' is the VCID and each *apid* is a digit from 0 to 2047, giving a desired source packet APID. The VCID, VCID list, and/or the APID list may be preceded by the keyword '`not`', which changes the filter sense from positive to negative. For '`ac ... frames ...`', the VCIDs in the filter presently may not be preceded by '`vc`'. For example, the packet filter '`vc0 1 2 3 59`' requests packets 1, 2, 3, and 59 on VC0. The packet filter '`vc0 not 1 2 3 59`' requests all packets on VC0 except for 1, 2, 3, and 59. Similarly, '`not vc0 1 2 3 59`' requests all packets on all VCs except for packets 1, 2, 3, and 59 only on VC0. Finally, '`not vc0 not 1 2 3 59`' requests all packets on all VCs except for all packets on VC0, but does request packets 1, 2, 3, and 59 on VC0.

After processing an acquire command, the controller will respond with '`error [message...]`' if an error occurred or with a message of the form '`ok` *handle* `[`*hostname port*`]`'. In this case, the *handle* is an opaque identifier that later may be used to adjust the filtering on the data connection. In response to '`ac tcp...`' commands, the *hostname* and *port* is the IP address that the user should connect to actually receive data. When the user successfully connects to the given host at the given port, the requested data will begin flowing immediately as it becomes available from the spacecraft. For UDP data flows, the datagrams will begin flowing immediately after the command response has been sent. Note that due to internal processing, datagrams may begin flowing before the response is received by the IGSE.

6 Telemetry Packet Grouping

ITOS provides optional handling of CCSDS packet groups whereby a packet group can be concatenated into a single large packet. A *packet group* is a set of packets of the same AppID related by the two-bit grouping flags in the CCSDS packet header. The newly formed packet can be assigned a new AppID. It is forwarded downstream to for unpacking and archiving just like all other packets.

Telemetry packet group concatenation is controlled by a configuration file, given by `gbl_tlm_pktgroup`. Each entry in the file consists of one line containing three integers: The incoming AppID, the outgoing AppID, and the expected length of the data field length for the outgoing packet. The third field currently is unused, but must be present, and we recommend that you set it to zero. Comments are introduced by the '#' character and continue to the end of the line. Here is an example:

```
# AppID in | AppID out | packet length out
     5          5             0
    10          2             0
```

If `gbl_tlm_pktgroup` is missing, or contains the null string, telemetry packet group handling is disabled. If it names a valid configuration file, then when ITOS extracts a telemetry packet from the transfer frame stream, it will look up the AppID in column one of the configuration file. If it finds the AppID listed, and the grouping flags in the packet header are set to 01 binary, it will begin accumulating a concatenation of the packet group. Subsequent packets of the same AppID should have grouping flags set to 00 binary until the last member of the group, which will carry a 10 binary in the grouping flags.

Unexpected grouping flag values (11 or 01 binary) or packet sequence errors during the accumulation of a group will cause the concatenation to be abandoned. Once the last packet in the group has been received, the new packet will be generated with the AppID given by the selected entry in the configuration file.

7 Command Wrappers

‘`acq`’
This prepends a CCSDS acquisition sequence of 144 bits of alternating ones and zeros (18 bytes of 0xaa) to the command.

‘`asist_swts`’
This wrapper transforms the outgoing command into an ASCII-hex coded command sequence surrounded by ASCII SFDUs that is suitable for sending to an NTGSE-based computer. This must be the outermost (last specified) wrapper when used.

‘`cltu`’
This generates a command link transmission unit (CLTU) from the command frame. This must be the innermost (first specified) wrapper when used.

‘`eos`’
This is an Earth Observing System (EOS) header adapted for use by the SMEX project, also known as LEO-T. The source is set from `gbl_nascom_src`, and the destination is set from `gbl_nascom_dest`.

‘`ipdu`’
This is the Inter-project Data Unit header as defined by Avtec PTP used by AstroRT for the Glast I&T facility. The source code is set from `gbl_nascom_src`, and the destination is set from `gbl_nascom_dest`. See *Avtec PTP for Windows User’s Guide, Ver.1.49, July 19, 2001 (TM -01-13), Appendix B tables B-1, B2, B3, B4* for details.

‘`itp`’
This prepends an ITP header to the command.

‘`nascom`’
This puts the outgoing command into a 4800-bit NASCOM block. It doesn’t work if the command is longer than will fit in the block. The NASCOM source code is set from `gbl_nascom_src`, and the destination is set from `gbl_nascom_dest`. Three formats are supported, selected by the desination code: shuttle, GN, and DSN. The ‘`WFF`’ code selects the GN format, the ‘`MDM`’ code selects the shuttle format, and everything else selects the DSN format.

‘`notf`’
This is a kludge that removes the transfer frame header so we can transmit packets. It must be the first wrapper to be useful. We are likely to remove this wrapper from the system in a future release when we have implemented a more rational way of transmitting packets.

‘`rand`’
This is the CCSDS standard randomization. See *CCSDS 201.0-B-3: Telecommand Part 1 – Channel Service. Blue Book. Issue 3. June 2000, section 3.3.1* for details.

‘`rtp`’
This prepends a Real-time Transport Protocol (RTP) header to the command. This is an internet standard header given in RFCxxxx, and is required for UDP/IP commanding over the NASA Integrated Services Network (NISN). We only implement the sequence numbering.

‘`scat`’
This transforms the outgoing command into an ASCII-hex coded command sequence suitable for sending to our SCAT-based front-end computer. This must be the outermost (last specified) wrapper when used.

'swts_txpkt'
: This wrapper transforms the outgoing command into an ASCII-hex coded command sequence that is suitable for sending to a SpaceWire Test Set-based computer. The result of this wrapper is a well-formed SWTS TXPKT command. This must be the outermost (last specified) wrapper when used.

'tsi' This is the command header TSI Telesys front-end system used by the Berkeley Ground System.

'uuencode'
: This runs the command through the uuencode UNIX application to convert the command to ASCII. This should be the outermost wrapper when sending commands through e-mail.

7.1 EOS Ground Message Header

The 24-byte EOS Ground Message Header looks like this:

```
+------+---+-----+------+---+------+------+-----+-----+-----+---+
| msg  | X | src | dest | X | date | scid | seq | s/w | msg | X |
| type |   | id  |  id  |   | time |      | num | ver | len |   |
|  1   | 1 |  1  |  1   | 1 |  7   |  2   |  2  |  2  |  2  | 4 |
+------+---+-----+------+---+------+------+-----+-----+-----+---+
```

where the fields are, in order:

field	bytes(s)	description
message type	0	Normally 3, (Command data message).
source ID	2	Set to the value given by gbl_nascom_src. Unused by GSFC SMEX and SMEX-heritage missions.
destination ID	3	Set to the value given by gbl_nascom_dest. Unused by GSFC SMEX and SMEX-heritage missions.
message date & time	5-11	The date and time the message was generated, UTC, in PB5 (option C) format. See below for details and reference PB5 Time Code 1982-05-27, *Aerospace Data System Standard, Part 5: Clock and Time Code Standard, Standard 5.6 Parallel Grouped Binary Time Code for Space and Ground Applications – PB5.*
spacecraft ID	12-13	
sequence number	14-15	A 16-bit upcounting sequence counter; unused by GSFC SMEX and SMEX-heritage missions.
software version	16-17	Set to zero; unused by GSFC SMEX and SMEX-heritage missions.
message length	18-19	The total message length in bytes, including the EOS ground message header.

The date/time field looks like this:

```
+------+-----+------+-------+------+------+
| flag | tjd | secs | msecs | usecs | fill |
|   1  |  14 |  17  |   10  |   10  |   4  |
+------+-----+------+-------+------+------+
```

where the fields are, in order:

field	bit(s)	description
flag	0	Always 1.
tjd	1-14	Truncated Julian Day – the number of days since October 10, 1995.
secs	15-31	Seconds of day.
msecs	0-9	Milliseconds of second.
usecs	10-19	Microseconds of millisecond.

7.2 IPDU Ground Message Header

The 32-byte IPDU Ground Message Header looks like this:

```
                                  +------+---+   +---+------+---+
                                  | head | X |   | X | test | X |
                                  | ver  |   |   |   |      |   |
                                  |  4   | 4 |   | 3 |  1   | 4 |
                                  +------+---+   +---+------+---+
                                  |          |   |             /
                                   \         .   |        ..../
                                    \       /    |       /
+-----+-----+---+----+-----+-----+---+-----+-----+------+---+-----+---+
| IPCU| IPDU| X | src| dest| mesg| X | head| msg | flags| X | scid| X |
| sync| len |   | id | id  | type|   | ver | time|      |   |     |   |
|  4  |  4  | 4 |  1 |  1  |  1  | 1 |  1  |  7  |   1  | 2 |  1  | 4 |
+-----+-----+---+----+-----+-----+---+-----+-----+------+---+-----+---+
                                           |     |
                                          /       \
                            ..............          ................
                           /                                        \
                          +------+-----+------+-------+-------+------+
                          | flag | tjd | secs | msecs | usecs | fill |
                          |   1  |  14 |  17  |   10  |   10  |   4  |
                          +------+-----+------+-------+-------+------+
```

where the fields are, in order:

field	bytes(s)	description
IPDU sync	0-3	IPDU sync code, 0x74C2472C.
IPDU len	4-7	Total message length in bytes including IPDU header.
source ID	12	Set to the value given by `gbl_nascom_src`.
destination ID	13	Set to the value given by `gbl_nascom_dest`.
message type	14	Set to 3, (Command data message).
header version	16	4 bits unsigned integer set to 1.

message time (GMT)	17-23	The date and time the message was generated, UTC, in NASA PB5 (option C) format. See below for details and reference PB5 Time Code 1982-05-27, *Aerospace Data System Standard, Part 5: Clock and Time Code Standard, Standard 5.6 Parallel Grouped Binary Time Code for Space and Ground Applications – PB5.* See time breakout below.
data indicator	25 bit 4	Operational = 0, Test data = 1; always set to 1.
spacecraft ID	28	Supplied from `gbl_spacecraftid`.

Where message time field are, in order as follows:

field	bit(s)	description
flag	0	Always 1.
tjd	1-14	Truncated Julian Day – the number of days since the epoch midnight October 10, 1995.
secs	15-31	Seconds of day.
msecs	0-9	Milliseconds of second.
usecs	10-19	Microseconds of millisecond.

Note: All unused, spare, fill or non-applicable fields for command data message type are set to zero (0).

7.3 NASCOM 4800-bit block header

The NASCOM 4800-bit block format depends on the destination ID; that is, different destinations expect blocks to have different formats. The basic format looks like this:

```
+--------+------+------+-----+-----+------+-------+
|  sync  | src  | dest | seq | fmt | scid | spare |\
+--------+------+------+-----+-----+------+-------+ \
|   24   |  8   |  8   |  3  |  5  |  8   |   8   |\ \
                                                    \ \
   ..................................................  \
   \   ................................................
    \ \
     \ \
      \ +------+-------+-------+-----+------+
       \| type | spare | flags | len | time | data ... |
        +------+-------+-------+-----+------+
        |  8   |   8   |   3   |  13 |  48  |
```

where the fields are, in order:

field	#bits	description
sync	24	sync pattern 0x627627
src	8	source ID, set from `gbl_nascom_src`.
dest	8	destination ID, set from `gbl_nascom_dest`.
seq	3	block sequence count
fmt	5	block format, always 9.
scid	8	spacecraft ID, set from `gbl_spacecraftid`.

type	8	message type, normally DSN throughput, 105 octal. If the destination is MDM (112) this field is set to MCC command message, 217 octal.
flags	3	always 010 binary.
len	13	message length
time	48	block time stamp, formatted as a PB4 time.

A PB4 time looks like this:

```
(bits)  1 2 3 ........ 11 12 ............... 38 39   48
        ---------------------------------------------------
        |0 0| day of year | milliseconds of day |0......0|
        ---------------------------------------------------
```

If the destination is WFF (96) or MDM (112), the 16-bit word at byte 596 (from 0) is set to 0x00ff.

If the destination is MDM (112), the spacecraft ID field, and following spare byte are set to all ones, the spare byte following the message type (byte 9 counting from 0) is set to the destination ID, and the data length is increased by 48 bits to account for a special command header, which is inserted at byte 18:

```
+--------+---------+----------+---------+
| msgnum | payload | cmd type | OU mode |
+--------+---------+----------+---------+
```

7.4 SpaceWire Test Set TXPKT command

The SpaceWire Test Set TXPKT command may be used to configure the SWTS to transmit a single packet one or more times.

```
swts txpkt <portNum> -p <spwAddr> <protocolID> <XX> <XX> <XX> ... <XX> [-L <loopCnt>]
```

where the fields are, in order:

Field	Description

portNum The physical SpaceWire port number, a value between 1 and 4 inclusive, from which the packet will be transmitted, set from `gbl_swts_txpkt_port_num`.

spwAddr	The SpaceWire address to which the packet will be routed, set from `gbl_swts_txpkt_spw_addr`.
protocolID	The SpaceWire protocol ID, set from `gbl_swts_txpkt_spw_protocol_id`.

loopCnt

The loop count is an optional parameter that directs the SWTS to transmit the packet a specified number of times, set from `gbl_swts_txpkt_loop_cnt`.

7.5 ASIST/NTGSE SFDU command

This command wrapper was developed to enable commands to be transmitted to a copy of the SpaceWire Test Set, or SWTS, that has been specially modified for the LRO project. The wrapper matches an interface that ASIST uses to communicate with the NTGSE.

```
CCSD3ZA0000100000165C7333IA0DEST00000011<cmdDest>C7333IA0LABL00000020<cmdLabl>C7333IA0
```

where the fields are, in order:

Field Description

cmdDest
The command destination, set from `gbl_asist_swts_cmd_dest`.

cmdLabl
The command label, set from `gbl_asist_swts_cmd_labl`.

cmdSsim
The command directive, set from `gbl_asist_swts_cmd_ssim`.

8 Command Control

8.1 Inter-Command Delay

The ITOS command subsystem can be set to insert a minimum delay between successive commands. Set the floating-point mnemonic `gbl_cmd_delay` to the desired delay in seconds.

8.2 Virtual Channel Selection

Commands my be transmitted on any command virtual channel. This is controlled by the STOL `vc` directive, which sets `gbl_vc`.

9 Command Networking

The `cmd_transmit` program can send commands using Internet Protocol (IP) sockets, over either the Transport Control Protocol (TCP) or User Datagram Protocol (UDP). It can acccept or initiate TCP connections, or send unicast or multicast UDP datagrams.

The use of mulitple, simultaneously established command streams is available when `gbl_cm_txport` is set to either '`client_tcp`' or '`server_tcp`'. The switch `gbl_cmd_portsw` activates the desired command stream via user input once the subsystem is enabled.

The command system defaults to a single stream convention to maintain the requirements of previous users. Therefore, the `gbl_cmd_num_ports` and `gbl_cmd_portsw` Globals default to 1 and 0 respectively by the database. The single stream User needs only set the standard `gbl_cmd_host` and `gbl_cmd_port` Globals prior to enabling the subsystem.

If mulitple streams are desired, the following must be completed prior to enabling the subsystem: GBL_CMD_NUM_CMDS set to the desired number of streams, corresponding `gbl_cmd_host[i]`.html,,,] and `gbl_cmd_port[i]`.html,,,] set for each command stream (the array index, i, is used to reflect the command stream selection and starts at 0; max. i = GBL_CMD_NUM_PORTS - 1), and the GBL_CMD_PORTSW Global set to activate the desired stream (streams are designated by their array index).

The advent of simultaneous command connections, mentioned above, caused the use of `gbl_cmd_host` and `gbl_cmd_port` to change. However, these changes should be transparent to single stream users. These Globals are now set either by the User (in single stream mode) or by software (multiple stream mode) to reflect the active command stream.

If `gbl_cm_txport` is set to '`server_tcp`', the system will listen on the IP port given by `gbl_cmd_port` for a connection. When the connection is broken, the program will reset and begin listening for a new connection. This cycle will continue until the command subsystem is shut down.

If `gbl_cm_txport` is set to '`client_tcp`', the system will initiate a connection to the host given by `gbl_cmd_host` and port in `gbl_cmd_port`. If the connection is broken the command subsystem must be shut down and restarted to re-establish it.

If `gbl_cm_txport` is set to '`udp`', the system will send datagrams to the host and port combination given by `gbl_cmd_host` and `gbl_cmd_port`. If the given host address is a class D address, the `cmd_transmit` program automatically will set the multicast time-to-live to 127 for global transmission.

If `gbl_cm_txport` is set to '`serial`', the system will send commands to the serial device named in `gbl_cmd_file` and the settings for the device are defined in `gbl_cmd_serial`.

Finally, if `gbl_cm_txport` is set to '`email`', the system will send commands in electronic mail to the address given by `gbl_cmd_host` with a Sender/Reply address listed in `gbl_cmd_host[1]`. Both should be a valid e-mail address with the latter being set to accomodate sending from behind a firewall. When using the '`email`' transport, the outermost wrapper should be '`uuencode`'. Note that `cmd_transmit` runs the `Mail` application to send commands through e-mail, so an SMTP server (`sendmail`) must be enabled on systems using this transport.

10 Command Packet Grouping

ITOS can create a group of command packets from a single long command packet. *Group* here means a set of packets with the same AppID related using the grouping flags in the CCSDS packet header. An AppID can be assigned to the commands in the group which is different from the AppID on the command from which the group was formed. The maximum length of commands in the group is user defined.

This functionality was added for the Swift BAT instrument, and can be used as an alternative to CCSDS command segmentation. The Swift spacecraft places a limit of about 60 bytes on the length of a command packet which can be passed to an instrument. The BAT instrument uses software with previous mission heritage, and for which commands longer than 60 bytes are defined. So the ability to split a command into a command group was added to ITOS so longer commands can be passed through the spacecraft bus to the instrument, which will re-assemble the packet group into a single packet the way ITOS can assemble telemetry packet groups into a single telemetry packet.

Command packet group creation is controlled by a configuration file, given by `gbl_cmd_pktgroup`. Each entry in the file consists of one line containing three integers: The incoming AppID, the outgoing AppID, and the maximum command packet data field length for outgoing packets. Comments are introduced by the '`#`' character and continue to the end of the line. Here is an example:

```
# Incoming Apid |   Outgoing Apid   | New Pkt Len
      5                 5              15
     10                 2               5
```

If `gbl_cmd_pktgroup` is missing, or contains the null string, command packet group creation is disabled. If it names a valid configuration file, then as ITOS generates a spacecraft command, it will look up the AppID in column one of the configuration file. If it finds the AppID listed, it will split the command packet into a group of command packets, with each command in the group given the AppID given by the second column of the selected entry. The length The data field of each command in the group, except possilby the last, will be the number of bytes given by the third column in the selected configuration file entry. If the original command is shorter than the maximum length, then it is sent with the new AppID but with it's grouping flags set to 11 binary.

Both the checksum given by `gbl_cmd_chksm_pkt` and any checksum defined in the database for the original (long) command are applied to each command in the group. For a two or four byte checksum, the packet length specified in the configuration file should be an even number. For SMEX-style CCSDS commands which contain two-byte secondary headers, the secondary header is duplicated in each command in the group. For regular CCSDS commands, any secondary header will appear only in the first packet in the group.